The Software Architecture of a Real-Time Battlefield Visualization Virtual Environment

Simon Julier[i], Rob King[ii], Brad Colbert[iii], Jim Durbin[iv] and Lawrence Rosenblum[v]

Abstract— **This paper describes the software architecture of Dragon, a real-time situational awareness virtual environment for battlefield visualization. Dragon receives data from a number of different sources and creates a single, coherent, and consistent three-dimensional display. We describe the problem of Battlefield Visualization and the challenges it imposes. We discuss the Dragon architecture, the rational for its design, and its performance in an actual application. The battlefield VR system is also suitable for similar civilian domains such as large-scale disaster relief and hostage rescue.**

***Index terms*—distributed VR; command and control; software architecture; interactive visualization.**

I. Introduction

Gaining a detailed understanding of the modern battle space is vital to the success of any military operation, both in terms of achieving objectives and minimizing civilian and military casualties. This understanding, also known as *situation awareness*, is used by Combat Operations Centers (COCs) in the complex task of directing the movement of assets and material over rugged terrain, day and night, in uncertain weather conditions, while taking account of possible enemy locations and activities. Most COCs receive vast amounts of data from many different systems and sources such as eyewitness reports, aerial and satellite photography, and a multitude of sensors such as sonar, radar, seismic, and infrared. Despite the availability of state-of-the-art sensors and data fusion technology, many command and control decisions typically use 2D displays and, for advanced planning, these are sometimes just paper maps with acetate overlays. Creating detailed maps and overlays is labor intensive and can take several hours to print and distribute. Given the rapid growth in available information and the increased tempo of warfare [Cebrowski-98], these methods no longer meet the needs of the Navy and Marine Corps.

The Virtual Reality Laboratory at the Naval Research Laboratory was one of the pioneers of the Responsive Workbench, having fabricated the first U.S. version in 1994 [Rosenblum et al., 1995] with applications in medicine and engineering design [Rosenblum et al., 1996]. More recently our thrust has been in developing Dragon for situational awareness on the Workbench and interoperably across platforms with differing display and interaction capabilities. Dragon is a prototype software platform for developing and investigating methods for visualizing and interacting with battlefield information in COCs.

Battlefield visualization is a special case of a planning task where a user must plan and coordinate multiple units within a complex and ever changing environment. It has a number of specialized needs (described in Section II) which impact both the software architecture and user interface design. We were unable to find a single system that could meet our requirements and it was necessary to develop a custom solution. This paper describes the development of this architecture and how it was tailored to meet our requirements. A companion paper describes the development and verification of the user interface through formative and summative user evaluations [Hix-99].

The structure of this paper is as follows. In Section II we describe the problem of battlefield visualization and outline the history of the Dragon system. The software architecture is described in Section III. The application of the Dragon system to a mine-clearing operational simulation is described in Section IV. The summary and conclusions are presented in Section V.

[i] NRL Virtual Reality Laboratory; contractor for ITT Systems & Sciences. Email: `julier@ait.nrl.navy.mil`

[ii] NRL Virtual Reality Laboratory; contractor for Wagner Associates. Email: `king@ait.nrl.navy.mil`

[iii] NRL Virtual Reality Laboratory; contractor for ITT Systems & Sciences. Email: `bcolbert@ait.nrl.navy.mil.`

[iv] Effectiveness of Navy Electronic Warfare Systems, NRL. Email: `durbin@nrl.navy.mil`

[v] NRL Virtual Reality Laboratory; Email: `rosenblu@ait.nrl.navy.mil`

II. Problem Statement

The NRL's Virtual Reality Laboratory was approached in 1996 by the Marine Corps Warfighting Laboratory to develop a prototype visualization system for the Hunter Warrior Advanced Warfighting Experiment (AWE). Hunter Warrior was the first in a series of experiments to explore technology and its role in supporting the Marines in the 21st century [Rosenblum et al., 1997].

The requirements for a battlespace visualization system include:

1. Interface to all of the appropriate military situation awareness, planning, and simulation systems. Each data source has its own characteristics, such as the types of data it provides, the queries that can be made, the protocols involved, and the update rates. The limitations and benefits of each type of data source must be preserved.
2. Display urban terrain, manmade objects, and objects and terrain under foliage.
3. Present a comprehensive and *timely* view of the environment. This includes displaying extremely large, high-resolution maps of the terrain and extremely large numbers of moving entities (on the order of hundreds or thousands).
4. Provide a dynamic range of resolution sufficient to track units ranging from aircraft carriers to individuals.
5. Visualize potential enemy and friendly courses of actions and neutral activity.
6. Support information filtering. It must be possible to display a selected subset of available information using appropriate display techniques.
7. Prioritize events and issue alarms.
8. Deliver information on demand. It should be possible to obtain any known information about a particular entity such as its fuel level or alert status.
9. Represent battlefield uncertainty. For example, certain data fusion procedures might be unable to unambiguously classify the identity of an entity.
10. Support multi-user collaboration.
11. Support multiple computer hardware configurations, ranging from low-end PCs to high-end Silicon Graphics (SGI) workstations.
12. Support multiple types of display devices, ranging from desktop displays to CAVE™-like devices.

These requirements are summarized in **Figure 1**, which shows a top-level architectural diagram of the Dragon system. There are three main issues: *interaction devices*, *display platforms*, and *information sources*. Information is received from many different types and kinds of disparate data sources including battlespace monitoring systems and this information must be fused into a single, consistent view of the environment. The user can view these data on one of a number of platforms and interact with it via many different types of input devices.

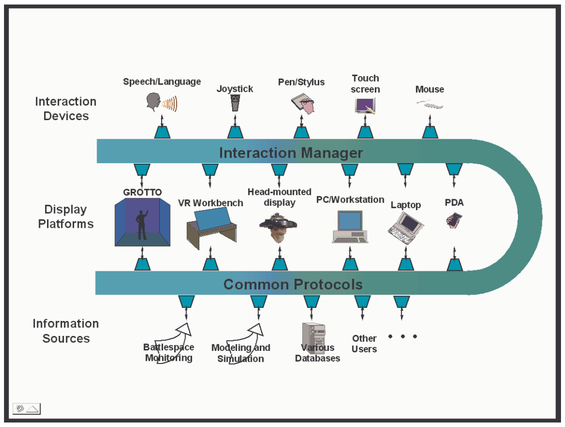
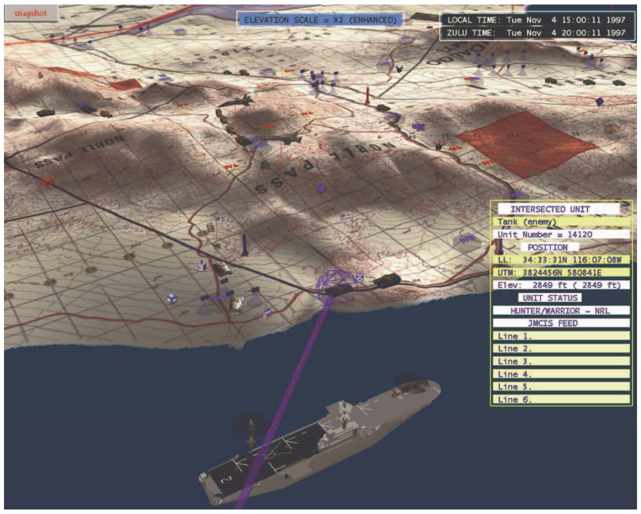


Figure 1: Role of the Architecture

At first sight, it appears that this architecture is very conventional and the visualization problems can be addressed by systems that have already been described in the literature. DIVE [Hagsand-96], for example, is a mature, multi-platform, multi-user collaborative VR system which can be extended by interpreted scripts (tcl/tk) or used as a library which could be linked to a custom application. Similarly, the Virtual Life Network [Padzic-97] has a set of "engines" which control aspects of the virtual environment such as object behavior or navigation. Each engine can be driven by an external interface through network connections. In theory, these engines could be driven directly from the interaction managers and external interfaces.

However, most existing VR systems and collaborative virtual environments [Churchill-98] are designed to support architectural features such as rooms, objects, and avatars as opposed to terrain, military vehicles, and planning symbology which are required to visualize the modern battlefield environment. For this reason, we developed our own architecture and software system.

To gain an appreciation of the application domain, Figure 2 shows the output from the Dragon system for the Hunter Warrior AWE [Durbin-98]. The environment consists of a terrain that is populated by a set of entities. The terrain must be an accurate representation of the physical conditions of the appropriate scenario. The geometry for the terrain is extracted from high-resolution digital terrain elevation data (DTED) and it is textured with an accurate, high-resolution map. Each entity represents a report from an external data system and can represent discrete military units or various types of planning symbology such as named areas of interest.



ELEVATION SCALE = X2 (ENHANCED)

LOCAL TIME: Tue Nov 4 15:00:11 1997
ZULU TIME: Tue Nov 4 20:00:11 1997

INTERSECTED UNIT
Tank (enemy)
Unit Number = 14120
POSITION
LL: 34:33:31N 116:07:08W
UTM: 3824456N 580841E
Elev: 2849 ft (2849 ft)
UNIT STATUS
HUNTER/WARRIOR – NRL
JMCIS FEED
Line 1.
Line 2.
Line 3.
Line 4.
Line 5.
Line 6.

Figure 2: Output from the Dragon System

III. ARCHITECTURE DESCRIPTION

The Dragon system, like many other complex modern software systems, is designed as a set of tightly coupled software modules. The current design, which is illustrated in Figure 3, consists of two major subsystems: the *Generic Entity Manager* (GEM) and *the Rendering Engine* (RE). GEM is responsible for collecting data from external data sources and expressing them in a common, standard representation. The RE is responsible for all user interaction; it draws the virtual environment, processes user input, and creates a set of requests and events that are directed back to GEM. The two subsystems interact by means of a pair of unidirectional event queues (that specifies the types of state changes that have occurred) and a shared entity database (that specifies the actual entity state information).

This design has a number of advantages. First, the clear separation of the entity manager from the renderer allows the two subsystems to operate asynchronously. The GEM is largely driven by the rate at which the external interfaces provide data. As will be described below, some of these systems provide many events per second whereas others provide data infrequently. The RE operates independently of these data rates, attempting to maintain constant and interactive frame rates (>10Hz stereo).

Second, the clear separation of the GEM and RE modules isolates possible interdependencies. All interactions between the two subsystems are accomplished via a clearly defined API and a strictly defined shared entity database. This means that different versions of the RE can be used with a given version of the GEM. For example, our current Rendering Engine (as discussed in Section III.C) is based on the IRIS Performer software libraries. However, any rendering system that conforms to the GEM-RE communications system (described in Section III.B) could be used.

Finally, this architecture encourages code reuse. Components of the RE have been transitioned to a parallel scientific visualization project [Kuo-99].

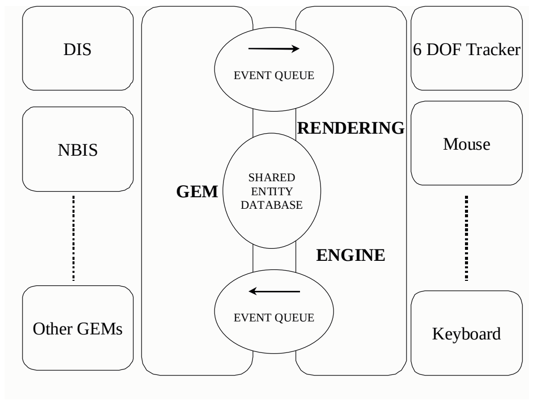


Figure 3: The Software Architecture

The following subsections describe the GEM, the RE, and the inter-module communication system.

A. Generic Entity Manager (GEM).

The GEM has two main components: an entity database, which stores information on all current entities in the environment, and a set of external systems interfaces. Each entity is created and controlled by an appropriate external systems interface. All entities are sub-classed from the same basic entity definition. To date we have implemented four interfaces: a persistent data store (or static data system), a Distributed Interactive Simulation (DIS) interface, a Joint Maritime Command Information System (NBIS) interface, and a remote GEM interface for multi-user collaboration.

1) Entity Definition

Each identifiable component of the environment, whether it is a solid object, such as a tank, or an abstract logical relationship, such as a named area of interest, is described as a type of entity. The entity definition must be sufficiently broad to describe all of the types of entities that the Dragon system will present. For a general environment this is a formidable task. Fortunately, the Department of Defense has addressed this need in the development of databases and communications systems.

Probably one of the best-known and most applicable classifications is that provided by DIS. DIS exhaustively enumerates most types and kinds of military assets encountered in a battlefield[1]. Recently, DIS has been superseded by the High-Level Architecture (HLA). HLA exploits recent developments in object oriented programming to provide a general, scalable and flexible framework for software development [DMSO-97].

At the time that we began the development of Dragon, a mature implementation of HLA was not available. Therefore, our entity description is a simplification of the HLA Federated Object Model incorporating the classification scheme from DIS. Specifically, the fields used in Dragon for each entity are listed in Table 1 below.

Identification
- *Source*
- *ID*
- *Type*

Graphical representation
- *Model name and scale*
- *Appearance*
- *Label*

Location

Attribute List

Ownership and Permissions

Table 1: Components of an entity

The meaning of each field is as follows:

- *Identification*. The Dragon system requires that each entity have a unique identity. An entity's identification consists of its *source*, its *ID* and its *type*. The source specifies the external data system providing the entity. The ID is the identification tag provided by the source system. Since different systems use different naming and labeling practices, GEM does not impose any kind of structure on this field. The type is the DIS enumeration describing the entity. It consists of six fields *(kind, domain, nationality, category, subcategory* and *specific.)* which together completely define the nationality, role and type of each entity. For example, kind=1, domain=1, nationality=225, category=1, subcategory=9, specific=1 is a prototype US AAI Rapid Deployment Force Light Tank Combined Arms Team/Lightweight Combat Vehicle.
- *Graphical Representation*. The graphical representation determines how a particular entity will be displayed to the user. The first field specifies the *model name* (typically its file name) and the *scaling* to be applied to the model after it has been loaded. Typically, the entity type determines its representation. A look up-table, defined by a configuration file, maps each entity type to a particular model name and scale. An entity model can contain a number of representations each reflecting a different entity state. For example, different models are used to represent an entity that is healthy, damaged, or destroyed. The *appearance* field determines the state to be shown. The *label* is an optional text label that can be attached to an entity. Supplemental information contains miscellaneous data, such as comments placed by a battlefield observer.
- *Location*. This specifies the position and orientation of the entity. Surprisingly enough, this is one of the most complicated parts of the entity definition. There are two reasons for this difficulty. First, although the Earth is extremely large it is *not* flat. On the scale of many battlefields – typically tens of kilometers on a side, the slight curvature of the Earth is just sufficient to introduce subtle but appreciable errors. The second problem is that there are literally dozens of different coordinate systems that can be used to describe the location of an entity. The coordinate systems we have implemented include geodetic (longitude and latitude to six or seven decimal places), Cartesian (earth-centered earth-fixed inertial) and the Universal Transverse Mercator. These different coordinate systems are converted to a Cartesian coordinate system. Even so, offsets must be applied to avoid truncation and rounding errors[2].
- *Attribute List*. Each entity can have a list of attributes attached to it. Each attribute is a data type, such as an integer or a string, that encapsulates an operational attribute of the entity.

[1] IEEE standard 1278.1- 1995.

[2] Performer only accepts single-precision floating point numbers for position. Both the Java 3D API and the latest release of the Performer libraries now provide mechanisms to accurately specify very large distances.

For example, the attribute might contain the fuel level of an aircraft.

- *Ownership and Permissions*. GEM provides a mechanism of supporting the policies of its external systems through the notion of ownership and permissions. This mechanism is drawn from the models used in DIVE [Hagsand-96], SPLINE [Waters-97], the MR Toolkit [Wang-95], and HLA [DMSO-97]. Each entity is owned by the external system that created it. Associated with each entity are a set of actions (such as delete, query or move) that are moderated by a set of access control lists and permissions. For example, a simulation system might not let a user, under normal circumstances, pick up and move an entity. However, during a set up phase when a scenario is created, the simulation system might grant the user entity movement capabilities.

The next subsections describe some of the external data systems and how the data is converted to the general entity format.

2) *Static*

The static data store is the simplest and one of the most useful data interfaces. Its name refers to the fact that it is a link to a static or persistent database. It offers the capability of storing a “snap shot” of the scenario at a particular moment in time that can then be loaded at a later date. The entities created by this system are not controlled or regulated by an external system, and can be used in the collaborative interface described later.

3) *DIS*

The DIS interface was developed to allow GEM to participate in Modular Semi-Automated Forces (ModSAF) simulation exercises as a “stealth viewer” – a device that can be used to explore and examine a virtual environment but cannot make any changes.

ModSAF is a multi-host virtual environment system for planning and shaping military activities [Courtmanche-95]. Using the DIS protocol mechanism, ModSAF contains models of the capabilities and behaviors of many different kinds and types of military assets. The Navy, the Army, and the Airforce use it extensively in their simulation systems.

To successfully interface with this system, GEM’s external interface had to act, as far as the simulation was concerned, as if it were another participant within the simulation. This was achieved by incorporating the vrLink libraries from MÄK Technologies. vrLink is an interface library that interprets each simulation data packet, builds a table of entities that currently exist in the simulation, and provides other simulation specific operations such as dead reckoning.

Since the connection was passive, the DIS system owned and controlled all of the entities. Users were able to select and query these objects, but they could not move them or change their state.

Due to its occasionally high data rates, DIS placed a number of demands on the design of Dragon. For example, whenever the interface to a running simulation is first opened, Dragon must create, within a matter of seconds, all of the entities populating the virtual environment. To maintain interactive rates despite this large number of events, asynchronous model loading schemes had to be developed. This is described in more detail in Section III.C.

4) *Network Battlefield Information System (NBIS)*

The Joint Maritime Command Information System (NBIS) is a widely used automated information system designed to support mission planning. It is composed of several mission applications integrated into a common XWindows-like operating environment and networked to support the sharing of raw sensor data, tracked information, and other types of databases. For Dragon we were interested in monitoring and displaying *track* information. Each track corresponds to a confirmed entity or object in the environment. Although NBIS uses its own methods for labeling and classifying tracks, we were able to make a one-to-one mapping from the NBIS to the DIS classification schemes by via a look-up table in a configuration file.

Unlike ModSAF, which is computer generated and is a constant stream of data packets, NBIS data arises from sensor and tracking systems as well as eyewitness reports. The data arrives infrequently and unpredictably (on the order of seconds to minutes between each data packet) and can contain ambiguities. For example, in the Hunter Warrior AWE, a great deal of NBIS data was provided by the Leatherneck system – a set of Marine observers equipped with an Apple Newton, a GPS receiver, and an Erickson radio who keyed their observations directly into the system. These observations were automatically routed and entered into NBIS. Messages could be incorrectly formatted or ambiguous. For example, a number of different observers viewing the same object could classify it as a different type of military entity or report a grossly different location.

In the Hunter Warrior AWE, the classification errors were predominantly and consistently of the misidentification and incorrect localization variety. We compensated for the misidentification by adjusting the NBIS to DIS mapping.

In general a visualization system like Dragon should not automatically correct for ambiguous or erroneous data as this might introduce false or misleading information. Rather, the system should inform the user that the potential for incorrect data is present and let the user decide the appropriate course of action.

5) *GEM to GEM*

Planning is an inherently collaborative activity and we developed a GEM to GEM interface to permit simple multi-user collaboration. The collaborative system was implemented using CAVERN as the transport layer [Leigh-97]. The collaboration system allows an arbitrary number of GEMs to be inter-connected.

Working on the assumption that collaborating GEMs will be connected to the same set of external systems, the collaborative interface is responsible for propagating a subset of entities that can be shared.

Furthermore, we have only implemented a subset of the full permissions-ownership capabilities. Currently, an entity can be owned by only one user who has permission to carry out all operations (This is not unlike the systems used in DIVE [Hagsand-96] and SPLINE [Waters-97]).

Initially, all shareable entities are not owned. To own an entity, a user selects it. When the entity becomes selected, this information is propagated to all other collaborating users who see the graphical representation of the entity (such as the color of a naming label) change to indicate that the entity is now owned by a particular user. The owner is now able to move or delete the entity. The entity's movements are reflected on the remote machines. Finally, the user deselects the object, thereby releasing ownership rights to it.

When GEM A wishes to collaborate with GEM B, A first requests permission to open a connection with B. Once the connection has been opened, both GEMS synchronize their local entity databases – GEM A receives a copy of GEM B's shareable entities and vice-versa. All subsequent collaborative interactions occur whenever a shareable entity is created, deleted, or changed. The state change is converted to an event message and state change data propagated on an as-needed basis. In our prototype scenario, we were networking only a very small number of GEMS (2-3) with very limited collaborative interaction (typically only 1-2 entities were manipulated at a time, generating at most 20 messages per second). Therefore, it was convenient to use the same format for expressing state changes as the static file format.

B. *GEM-RE Communication*

The GEM-RE communications subsystem provides the mechanism for transferring entity data to and from the RE. The state changes to the RE arise from the interactions of all of the external systems. The state changes from the RE consist of user initiated changes such as relocating an entity. For the types of systems we were dealing with, the greatest volume of data would flow from GEM to the RE. Furthermore, we could not afford to have either system sit and wait until it could acquire data from its counterpart.

The communications subsystem consists of three main parts: a GEM to RE event queue, a RE to GEM event queue, and a shared data pool. The interface uses a simple ownership toggling policy to coordinate interaction. First, GEM is granted ownership of the GEM-RE communications subsystem. Second, all of the events from the RE are processed, causing changes to occur in GEM's stored entity database. Each entry in the event queue specifies an entity and the type of interaction, such as move or delete. The data for the interaction is stored within the shared data pool. Third, all of the different external system interfaces are queried and all entity state changes since the last time step are calculated. These state changes are used to build the event queue to pass a list of instructions to the RE. Fourth, state change information on shareable objects is propagated out to the remote hosts. Finally, GEM grants the RE ownership of the communications system. The RE follows a similar pattern of operation, updating the state of all entities specified in the event queue. User invoked changes are placed into the RE to GEM event queue and the RE switches ownership back to GEM.

C. *The Rendering Engine*

The Rendering Engine (RE) is responsible for implementing the user interface and it performs two important roles – it manages the graphical display and processes the user inputs. The requirements for the RE are much more conventional than the GEM and, as such, there are fewer novel developments. However, the battlefield visualization application leads to two important demands: it is reconfigurable (can be used with multiple display devices) and should support interactive frame rates (>10Hz stereo) when large and complicated environments are being viewed.

1) Hardware Configurability

The Dragon system must operate with different suites of interaction devices and display platforms. This problem is not uncommon for the VR community and a number of general-purpose graphics libraries have been developed. One of the most extensive is the CAVE Library which is a comprehensive set of graphics system configuration tools, input device drivers and simple networking capabilities [Pape-97]. However, its architecture is *monolithic* – all of its capabilities are bundled into a single library and it was not possible, for example, to extend, adapt or modify the set of device drivers. Given these difficulties we developed our own graphical library, the vrLib, which builds directly on top of SGI's IRIS Performer libraries [Kuo-98].

The core of a vrLib-based program is an instance of a `vrApp` object that maintains all system configurations including the number and type of input devices as well as the configurations of the display screens. The abstract base class provides minimal functionality for loading and parsing configuration files. Subclasses of `vrApp` are responsible for interpreting the configuration commands and performing the actual graphics management. For example, the base class supports a parameterization of the viewing surface in terms of a user-specified number of windows with basic properties (such as size, pipe number and whether console input is enabled on a particular window). The concrete subclasses of `vrApp` interpret this information appropriately (for example, as the walls of a CAVE or a Workbench with ancillary displays).

2) Graphics Management

Roughly speaking, the dominant factor that influences the frame rate is the complexity of the scene (in terms of the number of triangles) that is being viewed by the user at any given time. Surprisingly enough, for our battlefield visualization applications the majority of the burden arises from the entities and *not* from the terrain. There are several reasons for this. First, the nature of our applications means that we only deal with compact and relatively "simple" environments at a single level of detail. With conventional polygon reduction algorithms, the terrain models typically use only about 10,000 triangles. In environments which are not populated (no entities), extremely high frame rates (>24Hz stereo) have been obtained without the need to use any kind of specialized terrain visualization techniques. Second, although the geometric description of each entity is fairly simple (for example, the model of an M1A1 tank has approximately 500 triangles), their large number means that they account for a significant proportion of the total polygon count. Finally, significant delays can occur when large numbers of models must be loaded in a short period of time. For example, a typical scenario requires over 100 different types of models. When Dragon is switched to a new scenario, it is possible that all of these models must be loaded in "one go" which can take on the order of ten seconds.

To address these difficulties, we used a *geometry library* and an *asynchronous model loading* strategy. The geometry library stores, in a parallel scene graph, a copy of the geometry of each model that has been loaded so far. When the RE receives a request to create a new entity, it checks the geometry library to see if the relevant model has already been loaded. If the model is present, its geometry is cloned into the scenegraph. If the geometry has not been loaded, a load request is scheduled with the asynchronous model loader. This is a separate process that loads the model and inserts it into the geometry library. Empirical tests suggest that these techniques approximately halved the initialization time and doubled the frame rate.

IV. Application

A recent application of the Dragon was as a support visualization system for the Joint Countermine Operational Simulation component of the Joint Counter Mine Advanced Technology Demonstrator (JCM). The purpose of JCM was to measure the effectiveness of novel techniques for mine clearance in combat situations. Sets of simulated exercises were conducted at North Carolina and the Dragon system, through its DIS interface, was used as a viewer to aid in the analysis of the archived data. This application is of particular interest to the architecture because it is extremely complicated and involves a large number of independent entities (more than 220) of many different types (over 100 different models were used). These entities were in constant motion and events such as explosions and smoke had to be depicted.

Throughout the exercise the system maintained a rate of at least 10 frames per second, thus meeting all of its requirements.

V. Summary and Conclusions

This paper has described the design of a software architecture for a real-time battlefield visualization system. Battlefield visualization has a number of unique requirements that directly impact the design of the architecture. For example, the need to interact with multiple disparate data systems lead to the development of the GEM and the RE.

Although this design has successfully met its requirements, a number of shortcomings and areas for future research work have become apparent:

1. Although the DIS classification scheme is extremely powerful, standardized, and general for describing military assets and entities, it is not capable of describing *all* types of environmental features that we might want to visualize, such as weather effects. Therefore, the entity classification system must be expanded.
2. The notion of an external interface is extremely useful but limiting. Many types of higher order object behavior, such as dynamic systems or windows systems, will own objects with their own types and behaviors. Therefore, the external interface will be generalized to an "actor" – any kind of object that can own and manipulate entities.
3. The system architecture will be enhanced to provide greater scalability and extensibility. The concept of independent modules will be pursued aggressively. Using the framework from Bamboo [Watsen-98], Dragon will be decomposed into a large number of interacting modules that can be dynamically loaded and unloaded at run-time. This will give a highly reconfigurable system that can, for example, load and unload external interfaces on demand.

VI. Acknowledgements

This research was supported by the Office of Naval Research, Arlington, Virginia.

VII. References

[Cebrowski-98] Vice Admiral A. K. Cebrowski and J. J. Garstka, "Network-Centric Warfare: Its Origin and Future", *Naval Institute Proceedings*, January 1998.

[Churchill-98], E. F. Churchill and D. Snowdon, "Collaborative Virtual Environments: An Introductory Review of Issues and Systems", *Virtual Reality*, Vol 3.1, pp 3–15, 1998.

[Courtmanche-95] A. J. Courtmanche, A. Ceranowicz, "ModSAF Development Status", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Univ. of Central Florida, pp 3-13, Orlando, Fl., 1995.

[DMSO-97] Defense Modeling and Simulation Office, "HLA Interface Specification", Version 1.1, 12 February, 1997.

[Hagsand-96] O. Hagsand, "Interactive Multi-user VEs in the DIVE System", *IEEE Multimedia*, Vol 3.1, pp 30–39, November, 1996.

[Hix-99] D. Hix, J. E. Swan II, J. L. Gabard, M. McGee, J. Durbin and T. King, "User-Centered Design and Evaluation of a Real-Time Battlefield Visualization Virtual Environment", *To be presented at the IEEE Virtual Reality '99 Conference, Houston, TX*.

[Kuo-99] E. Kuo, M. Lanzagorta, R. Rosenberg and S. Julier "A Report on VR Scientific *To be presented a the IEEE Virtual Reality '99 Conference, Houston, TX*.

[Leigh-97] J. Leigh, A. E. Johnson, T. A. DeFanti, "CAVERN: A Distributed Architecture for Supporting Scalable Persistence and Interoperability in Collaborative Virtual Environments" *Journal of Virtual Reality Research, Development and Applications*, Vol 2.2, pp. 217–237, December, 1997.

[Padzic-97] I.Pandzic, T.Capin, E.Lee, N.Magnenat Thalmann and D.Thalmann, "A Flexible Architecture for Virtual Humans in Networked Collaborative Virtual Environments", *Proc. Eurographics '97*, pp.177-188, 1997.

[Pape-97] D. Pape, C. Cruz-Neira and M. Czernuszenko, "The CAVE Library version 2.6 User's Guide", http://evlweb.eecs.uic.edu/pape/CAVE/prog/CAVEGuide.html

[Rosenblum et al., 1995] Rosenblum, L., S. Bryson, and S. Feiner, "Virtual reality unbound," IEEE CG&A, Vol. 15, No. 5, Sept. 1995, pp. 19-21.

[Rosenblum et al., 1996] Rosenblum, L. J., J. Durbin, L. Sibert, D. Tate, J. Templeman, U. Obeysekare, J. Agrawaal, D. Fasulo, T. Myers, G. Newton, A. Shalev, "Shipboard VR: from damage control to design", *IEEE Computer Graphics and Applications*, Vol. 15, No. 6, Nov. 1996.

[Rosenblum et al., 1997] Rosenblum, L.J., J. Durbin, R. Doyle, and D. Tate, "Situational Awareness Using the VR Responsive Workbench," *IEEE Computer Graphics and Applications*, Vol. 16, No. 4, July, 1997.

[Wang-95] Q. Wang, M. Green and C. Shaw, "EM – An Environment Manager for Building Networked Virtual Environments", *IEEE Virtual Reality Annual International Symposium*, Research Triangle Park, NC, pp 11-18, March 1995.

[Waters-97] R. C. Waters and J. W. Barrus, "The Rise of Shared Virtual Environments", *IEEE Spectrum*, March, 1997

[Watsen-98] K. Watsen and M. Zyda, "Bamboo - A Portable System for Dynamically Extensible, Real-time, Networked, Virtual Environments", IEEE Virtual Reality Annual International Symposium (VRAIS'98), Atlanta, Georgia, 1998.